RRDtrace: Long-term Raw Network Traffic Recording using Fixed-size Storage

Antonis Papadogiannakis,[*] Michalis Polychronakis,[†] and Evangelos P. Markatos[*]

[*]Institute of Computer Science, Foundation for Research and Technology – Hellas, Heraklion, Greece
{papadog,markatos}@ics.forth.gr
[†]Computer Science Department, Columbia University, New York, USA
mikepo@cs.columbia.edu

***Abstract*—Recording raw network traffic for long-term periods can be extremely beneficial for a multitude of monitoring and security applications. However, storing all traffic of high volume networks is infeasible even for short-term periods due to the increased storage requirements. Traditional approaches for data reduction like aggregation and sampling either require knowing the traffic features of interest in advance, or reduce the traffic volume by selecting a representative set of packets uniformly over the collecting period. In this work we present *RRDtrace*, a technique for storing full-payload packets for arbitrary long periods using fixed-size storage. RRDtrace divides time into intervals and retains a larger number of packets for most recent intervals. As traffic ages, an aging daemon is responsible for dynamically reducing its storage space by keeping smaller representative groups of packets, adapting the sampling rate accordingly. We evaluate the accuracy of RRDtrace on inferring the flow size distribution, distribution of traffic among applications, and percentage of malicious population. Our results show that RRDtrace can accurately estimate these properties using the suitable sampling strategy, some of them for arbitrary long time and others only for a recent period.**

I. Introduction

Live traffic monitoring systems capture and process packets in real time. Regardless of the particular use, captured packets are usually discarded once processed. However, *recording* the raw network traffic to disk for long-term periods can be very useful for a multitude of applications, such as troubleshooting network problems and measuring traffic trends or observing the historical evolution of the traffic. Moreover, while the Internet evolves over the years, new applications and more security breaches appear. Thus, long-term recording of Internet traffic can significantly contribute to better analyze and understand the Internet evolution.

Network traffic recording is also critical for many security purposes. Anomaly detection techniques require a long-term baseline of past traffic to build profiles for normal traffic and users. Postmortem forensics analysis is also based on past traffic to identify malicious activities that happened before the time that an attack is detected. For instance, looking back in time can help us to identify how the attackers compromised a system, what they did, and find out which data have been exposed to them. Moreover, lawful interception and data retention have been enforced recently by many national regulations to enrich crime evidence by reconstructing past VoIP calls or other kinds of network-based communications.

When new vulnerabilities and attack signatures for Network Intrusion Detection Systems (NIDS) are released, long-term recording of network traffic allows to identify past attacks and compromised systems that otherwise would go undetected. Also, it is common practice to test new NIDS signatures using past traffic to eliminate false positives. NIDS and other passive monitoring applications are trained, tuned, and properly configured based on recorded traffic from the network in which they will be deployed. Packet traces are also commonly used for benchmarking network monitoring applications and can be replayed in different rates using tools like `tcpreplay` [1].

Unfortunately, recording all traffic in high volume networks is impossible even for short-term periods, due to the high storage needs. For instance, a network with 300 Mbit/sec average load requires about 3.2 TB of storage for recording one day's traffic. Thus, the limited storage resources of a commodity PC allow for storing hours or maybe a few days of traffic in the best case. However, recording the network traffic for long-term periods using a reasonable amount of storage would be extremely beneficial for all applications mentioned above.

Storing only the first few bytes from each packet, which typically corresponds to protocol headers, can reduce the required storage and increase data retention [2]. However, monitoring applications that need to inspect both the headers and the payload of the packets, a process widely known as *deep packet inspection* [3], cannot operate with header-only traces. Two traditional approaches for data reduction are aggregation and sampling. Aggregation is effective when the traffic's features of interest are known in advance, while sampling techniques select a representative group of packets uniformly over time. The sampling rate is an important parameter for the accuracy on inferring various network metrics. Higher sampling rates result to better accuracy but require more storage space, and thus retention is reduced when using fixed-size storage. On the other hand, lower sampling rates increase data retention but inevitably reduce the accuracy of many applications.

In this paper we present *RRDtrace*, a technique for storing packets for long-term periods in fixed-size storage, inspired by the popular RRDtool [4]. We choose to store full-payload packet traces, which provide a rich source of information

suitable for all kinds of analyses, from coarse-grained measurements of network properties to fine-grained operations like deep packet inspection. *RRDtrace* divides time into intervals and retains more detail for more recent intervals, i.e, allocates more storage to recent time intervals and less storage to older time intervals. Also, older time intervals become longer than more recent ones. RRDtrace is based on an *aging* mechanism that dynamically reduces the space occupied by the data of a time interval as it ages, by keeping only a subset of the packets of that interval using sampling. Thus, as a time interval gets older, the sampling rate for storing its data decreases.

Many sampling techniques have been extensively studied for applications like traffic accounting, billing, and measurements like heavy-hitters identification and flow size estimation. However, the applicability of sampling techniques in other passive monitoring applications like traffic classification and intrusion detection has not received the same attention. Our study attempts to answer the following questions:

- *Which sampling strategies should be used to select a useful subset of packets when reducing the storage space that will allow us to infer as many as possible desirable properties from the trace? Which strategies are suitable for which properties?*
- *How much back in time can we go, i.e., what is the lowest sampling rate that still allows us to infer desirable properties from an RRDtrace with acceptable accuracy?*

To answer these questions, we evaluate the impact of three different sampling strategies with decreasing sampling rates on inferring desirable network properties using a large trace of real traffic. Our results indicate that RRDtrace using flow sampling can accurately estimate flow size distribution and distribution of flows among applications regardless of the sampling rate. Average flow size and percentage of traffic per application are estimated more accurately in recent time intervals. For estimating the percentage of malicious hosts and flows, reduction of traffic volume using a per-flow cutoff provides the more accurate estimates for recent intervals. Random packet sampling performs well only for few of the examined properties. Compared to a constant sampling rate strategy, RRDtrace can store traffic for arbitrary long time periods and offers higher accuracy for more recent traffic.

In summary, the main contributions of this work are:

- We propose and implement *RRDtrace*, a new approach to record network traffic for arbitrary long periods using fixed-size storage space. Our approach is based on reducing the sampling rate as traffic ages.
- We evaluate the impact of different sampling strategies and decreasing sampling rates on monitoring and security applications that can take advantage of RRDtrace.

The rest of the paper is organized as follows: Section II summarizes related work. In Section III we describe the storage allocation algorithm, sampling strategies and possible applications of RRDtrace. In Section IV we compare the retention of RRDtrace with other approaches when deployed to an operational network. Section V presents the experimental evaluation of RRDtrace using real traffic with monitoring and security applications. Finally, section VI concludes the paper.

II. Related Work

A first approach to increase retention when storing network traffic is to keep less data per packet. A common choice is to store only the first few bytes of each packet, which typically correspond to protocol headers. Solely from protocol headers, monitoring applications can infer useful information and network metrics, while this approach can reduce significantly the storage space [2] and thus increase retention. However, monitoring applications like accurate traffic classification, as well as security applications usually perform deep packet inspection operations, which require both the headers as well as the payload of each packet [3]. For instance, peer-to-peer and multimedia traffic identification [5] and NIDS [6], [7] employ protocol parsing and advanced pattern matching operations to identify application-specific strings or attack signatures in the packet payload. Thus, these applications cannot operate with header-only traces. Moreover, even with this significant reduction in storage requirements, retention time is still limited.

Another approach for efficient traffic recording is applied in the Time Machine system [8], where only the first N bytes of each flow are recorded based on a *per-flow cutoff*. This approach leverages the heavy tailed distribution of flow sizes that is commonly found in Internet traffic, since most of the traffic in a high volume network comes from just a few flows. Therefore, most of the flows will not be affected by the cutoff and will be fully recorded, while recording only the beginning of a few large flows leads to significant savings in disk space. However, this technique cannot accurately estimate network metrics like total traffic volume and flow sizes. Furthermore, Time Machine stores approximately the same amount of traffic per day, and thus inevitably can store traffic for a few days only and then will delete the oldest traffic.

Another solution that is commonly used to retain information about network usage in high volume networks for long-term periods is to maintain higher-level abstractions of the network traffic [9]–[11] or store aggregated data like NetFlow records [12]. Storing aggregated data instead of network packets can reduce dramatically the required disk space, while other higher-level abstractions can be used with fixed-size storage. However, such data formats limit significantly their usefulness. They can be adequate only for specific applications if the features of interest are known a priori. Any packet-level information will be lost, so many applications and deep packet inspection techniques do not work with aggregated traffic summaries. On the other hand, full-payload packet traces offer a rich source of information and allow for fine-grained analysis.

RRDtool [4] employs a Round Robin Database to store time-series data for very long periods in fixed-size storage using data aggregation. This feature of RRDtool has made it a popular choice for storing and visualising time-series data like temperatures, CPU load and network metrics like bandwidth,

delays, packet loss and many other. RRDtool is based on an aging process using a consolidation function (usually average) to consolidate multiple primary data points to form a single consolidated data point. Therefore, older data will have less detail but will be representative for the corresponding time periods. In this paper we aim to utilize the RRD properties to store network packets for long-term periods in fixed-size storage, using suitable aging mechanisms.

Cooke et al. [13] present a multi-format data storage technique that works with fixed storage and fixed time. First, packets are stored, and later on they are aggregated and transformed into flows as they age. Flows are finally aggregated into counters. Storage allocation algorithms divide the available storage between these different aggregation levels. The main shortcoming of this technique is that fine-grained analysis cannot be performed in old data, e.g., find possible undetected attacks or identify peer-to-peer and multimedia traffic using flow information. Moreover, having different data formats over time makes the analysis more difficult than having always the same data format.

Instead of storing actual packets, payload attribution techniques [14] store compressed digests of packet payloads. Based on an excerpt of a given packet payload, these techniques indicate the presence of packets that contained this exact payload and their source, destination and time of appearance on the network. Though, the actual payloads of the stored packets cannot be inferred. Such techniques are useful for forensics analysis and some security applications. Spring and Wetherall [15] present an algorithm for traffic compression by identifying and eliminating redundancy. Compression can effectively reduce the storage for protocols and applications with high redundancy.

Anderson et al. [16] present tools for recording packets at kernel-level to provide bulk capture at high rates. Hyperion [17] employs a write-optimized stream file system for high speed storage and Bloom filters for indexing stream data. Gigascope [18] is a stream database which offers an SQL-like language for queries on the packet stream, but does not focus on long-term archival.

Packet sampling is very common in high-end routers, where processing and storage resources are limited, for recording aggregated sampled traffic statistics like sampled NetFlow [19]. Several sampling strategies have been proposed, which are currently being standardized by the Packet Sampling Working Group of IETF [20]. The choice of a suitable strategy depends on traffic characteristics or on statistics needed to be inferred.

The basic idea of RRDtrace is that it adapts the sampling rate according to the time that traffic was captured. Rate adaptive sampling has been proposed for dealing with traffic load variability. Adaptive NetFlow [19] uses traffic rate prediction techniques to adjust properly the sampling rate. Drobisz and Christensen [21] present an adaptive scheme based on CPU utilization and packet interarrival times. Choi et al. [22] determine the sampling probability adaptively according to traffic dynamics to accurate traffic load estimation. Hernandez et al. [23] use a predictive approach to anticipate load variations and adjust accordingly the sampling interval to meet sampling volume constraints. Similarly, rate constrained sampling approaches select a specified number of packets during a measurement interval. The method proposed by Duffield et al. [24] works under strict resource constraints by sampling into a buffer of fixed size. All these approaches adapt the sampling rate based on traffic load, while RRDtrace adapts the sampling rate according to how old the stored traffic is, in order to provide better accuracy for the most recent traffic.

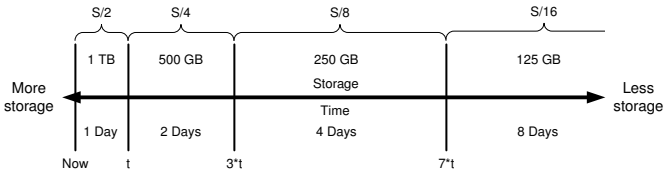

Fig. 1. Storage allocation in RRDtrace for S=2 TB.

Brauckhoff et al. [25] examine the impact of packet sampling on anomaly detection metrics for the Blaster worm outbreak. Blaster uses random scanning in TCP port 135, so it can be detected using flow counters. However, flow counters are heavily affected from packet sampling. While packet and byte counters are not affected from sampling, they cannot detect Blaster anomalies. The flow entropy metric is shown to be more robust to packet sampling than flow counters. Mai et al. [26] examine the performance of volume and port scan anomaly detection methods with sampled data using four different strategies. The results show that all the sampling strategies significantly degrade the performance of the detection algorithms. Among the four sampling schemes, random flow sampling introduces the least amount of distortion. Smart sampling [27] and sample-and-hold [28] are less resource intensive than random flow sampling, but perform poorly in the context of anomaly detection, since they miss small flows that are often related to attacks.

III. Our Approach: *RRDtrace*

Our approach, called *RRDtrace*, is inspired from the properties found in round robin databases. It aims to store full-payload packets for long-term periods in fixed-size storage. RRDtrace divides the time into unequal intervals and retains more packets from recent intervals, while keeping smaller subsets of packets from older intervals. Older time intervals are longer and utilize less storage. The duration of time intervals and how the available storage is assigned to them can be defined either by the users, according to the network in which RRDtrace will be deployed, or automatically by RRDtrace.

A typical example of storage allocation in RRDtrace is shown in Figure 1. We assume that the available storage for RRDtrace is $S = 2$ TB. We select the initial time interval t_0 to be one day and we assign the half storage (1 TB) to it. The next time interval t_1 is twice as large as t_0 with the half storage of t_0, i.e., t_1 is two days long with 500 GB storage. Thus, in t_1 (days 2–3), 1 out of 4 packets that were initially stored is selected to remain in the trace. Each subsequent time

interval is two times larger and has half the storage than its preceding one.

In this storage allocation algorithm different initial time intervals t_0 can be defined, occupying the half of the available storage. All the next intervals are formed based on t_0 and available storage S. In case that the traffic volume in t_0 is less than $S/2$, all packets in this interval can be stored. Else, packet sampling is imposed from the first time interval. An other option is to let RRDtrace to select the first interval t_0 in a way that all the packets during this interval are stored in the corresponding storage (with no sampling). Then, t_0 will be the time interval with traffic volume equal to $S/2$. This approach works well when the traffic volume in t_0 intervals does not vary significantly.

When a t_0 period passes, an *aging* daemon is responsible to appropriately reduce the storage used in each time interval. For instance, the number of packets stored during the last t_0 will be reduced by 25%, and similarly with the next intervals in order to conform with the storage allocation scheme described above. The aging daemon reduces the storage capacity in each interval by selecting a representative group of packets with the appropriate sampling rate. The packet selection strategy is an important parameter for the usefulness of RRDtrace.

We suggest the use of sampling instead of aggregation for two reasons. First, data is retained in the same format, which is very convenient for analysis and processing by existing applications. Moreover, aggregation requires knowledge of the traffic's features of interest in advance, whereas sampling allows the retention of arbitrary detail while at the same time reducing data volumes.

A. Sampling Strategies

Since RRDtrace may be used by multiple applications, different sampling strategies may be suitable for different applications. We have implemented three sampling strategies to evaluate their effectiveness using several monitoring applications. Each sampling strategy defines the way that k packets should be selected out of the total N packets in a time interval (sampling rate $s = k/N$), to respectively reduce the storage. We consider that a sampling rate has a similar effect in packets and storage reduction.

1) Packet Sampling: The simplest strategy to select k out of N packets is systematic count-based sampling, i.e., selecting one every N/k packets. However, systematic sampling is vulnerable to bias errors due to synchronisation with periodic patterns in the traffic and can be predicted.

Random packet sampling avoids the potential problems of systematic sampling. We choose to implement stratified random sampling. In this technique, the N packets are divided to k equal groups (with size of N/k packets) and one packet from each group is randomly selected. In systematic count-based sampling the first packet of each group would be always selected.

2) Flow Sampling: Research works by Hohn and Veitch [29] and Duffield et al. [30] have shown that packet sampling is inaccurate for the inference of flow statistics such as the original flow size distribution. For instance, it is easy to miss completely the short flows. Flow sampling has been proposed as an alternative to overcome the limitations of packet sampling. Hohn and Veitch [29] show that flow sampling improves the accuracy in flow statistics inference.

When a flow is selected, all the packets that belong to this flow are stored, while from an unselected flow no packets are stored. Flow sampling approaches for forming flow records focus mostly on selecting large flows, which has a larger impact to billing and accounting applications. So, non-uniform flow sampling techniques, like smart sampling [27] and sample-and-hold [28], have been proposed for accurate estimation of heavy hitters. These techniques give higher probabilities in large flows to be selected and form flow records.

In our case, we aim to select a representative group of flows for applications like traffic classification, building profiles, and security applications. Thus, we choose a uniform flow sampling approach. Random flow sampling with sampling rate s could be used. Similarly, hash-based sampling could be performed, using a hash function over the 5-tuple which defines a flow and then selects k out of the possible N hash values. However, these approaches do not guarantee that the selected flows will result to k out of N packets selection, and to the desirable storage reduction, due to the heavy-tailed distribution of flow sizes. Therefore, hash-based and simple random flow sampling, as well as smart and sample-and-hold sampling strategies, cannot accurately reduce the storage.

We need to specify a flow sampling scheme that selects l flows out of the M total flows in a time interval, with k packets in total. This flow sampling scheme works as follows: First we classify packets into flows. During the classification, we maintain an indexing table with the flows sorted based on their size and a histogram with flow sizes. Then, we randomly select one flow at a time, with a size of x_i packets, while $\sum x_i < k$ stands. Only flows with size less than $k - \sum x_i$ packets that have not been selected so far, are candidates for selection. These flows can be easily found using the indexing table and the histogram with flow sizes. Assuming that we have F flows with size less than $k - \sum x_i$ packets that have not been selected before, a random number from 1 to F is used to select the corresponding flow from the indexing table. The selected flows are marked and removed from the indexing table and flow size histogram. The selection process ends when l flows with $\sum_{i=0}^{l} x_i = k$ have been selected. Finally, the packets from the selected flows are written to disk, with a second pass in the trace, in respect to the order that they have been received.

3) Per-Flow Cutoff: Our third strategy for selecting representative packets is to use a per-flow cutoff, i.e., select always the first C packets of each flow. Time Machine [8] uses a statically user configured per-flow cutoff to limit the amount of traffic that will be stored. On the other hand, RRDtrace reduces the amount of traffic that will be stored according to the time interval that the traffic belongs to, thus different cutoffs are applied to different time intervals. As traffic ages, the per-flow cutoff will be properly reduced.

We implemented an algorithm that selects a per-flow cutoff C in a way that k packets are selected out of the total N packets. The algorithm is based on a histogram of aggregated statistics. In the first step we classify packets in flows. During this classification, we also maintain a table which indicates the number of flows that exceed each flow size. For instance, the position i of the table, $t[i]$, will contain the number of flows that have at least i packets. When the i_{th} packet of a flow is classified, $t[i]$ will be incremented by one.

Using this table, we can find the number of packets that correspond to a specific per-flow cutoff x from $\sum_{i=0}^{x} t[i]$. The selected cutoff C will be the largest position in the table that $\sum_{i=0}^{C} t[i] \leq k$ will be valid. In the second step of the algorithm, having the proper cutoff C, packets are classified again into flows and each packet is selected only if its position in the flow is less than C. Otherwise, the packet is not stored in the new file.

This per-flow cutoff strategy selects k packets in total from all the flows that appear in a time interval. Thus, it can accurately estimate the number of flows but not their size. Its main advantage is that the trace will contain the first packets from all the flows, so it will be suitable for security applications, e.g., port scan and intrusion detection, but not for traffic classification and accounting applications.

B. Implementation

RRDtrace is implemented using two separate threads: the capture and aging daemons. The *capture daemon* uses libpcap [31] to capture packets for the t_0 interval, impose sampling, if needed, in t_0 and initially store the packets in a memory buffer. When the memory buffer becomes full, the packets are written to disk. Separate files are used for each t_0.

The *aging daemon* is responsible for reducing the storage as traffic ages. After each t_0, it reads packets from the files of each interval, imposes the new sampling rates and writes the selected packets to the updated files. The two threads do not access the disk concurrently to improve disk's performance. Thus, the aging daemon runs only when the capture daemon writes packets to the memory buffer.

C. Applications of RRDtrace

We focus on using RRDtrace for the following two classes of possible applications:

1) *Study the historical evolution of traffic:* Using RRDtrace we aim to infer the distribution of traffic among different applications, the distribution of flows sizes, the percentage of the malicious population and how all these change over the years.
2) *Security applications:*
   a) Building profiles for normal traffic patterns based on RRDtrace to be used by anomaly detection metrics.
   b) Forensics analysis, which often requires the reconstruction of past streams for lawful interception or inspecting past traffic from suspicious or compromised hosts to identify more malicious operations or sensitive data exposed to attackers.

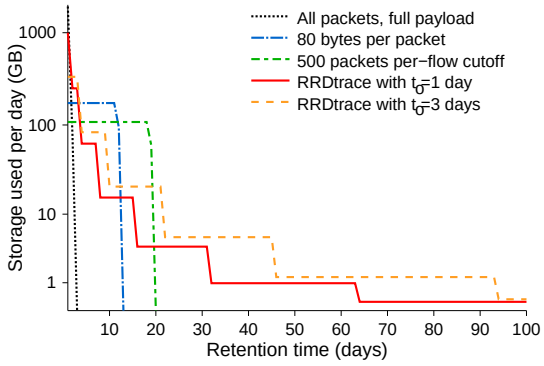

Fig. 2. Retention time and storage utilization for RRDtrace and other approaches with 2 TB of available storage.

   c) Intrusion detection in past traffic, for training new signatures to eliminate false positives, for detecting past attacks that were using a recently disclosed vulnerability or for estimating the percentage of infected hosts.

IV. Retention Study

We examine the operation of RRDtrace and compare its retention with other approaches by capturing and storing the traffic in the access link of an educational network. The average traffic load in the network is 178 Mbps with total traffic 1.92 TB/day on average. Assuming we have 2 TB available storage, t_0 should be set to 12.5 hours in order to store all packets during this interval in 1 TB. After t_0, for the next 25 hours, 25% of these packets will be stored in 500 TB.

Figure 2 presents the retention and the corresponding storage used per day for full-payload packet recording, headers-only recording (80 bytes per packet), when recording the first 500 packets per-flow and when using RRDtrace with $t_0 = 1$ day and $t_0 = 3$ days.

Since the daily traffic volume in the network is 1.92 TB, we can store all the packets with full payload for 25 hours only in the 2 TB storage. When capturing and storing only 80 bytes per packet, 173.22 GB are required per day, which results to 11.55 days retention. Applying a per-flow cutoff is a more effective approach, due to the heavy-tailed distribution of flow sizes. Using a cutoff of 500 packets per flow results to 107.76 GB/day stored and 18.56 days retention. A cutoff of 100 packets per flow results to 67.86 GB/day and 29.47 days retention.

On the other hand, retention in RRDtrace can be arbitrary large. Figure 2 shows the storage allocation in RRDtrace for the first 100 days using two different values of t_0. For $t_0 = 1$ day, 1 TB will be used for the last day's traffic, 15.6 GB/day for 8–15 days ago and 976 MB/day for 32–63 days ago. Selecting 976 MB from the total 1.92 TB daily traffic implies 0.05% sampling rate. For one year ago, 15.3 MB/day traffic will be available. When $t_0 = 3$ days, 333 GB/day will be used for the three last days. For 10–20 days ago, 20.83 GB/day will be stored, which implies 1.1% sampling rate. For one year ago, 81.4 MB/day traffic will be stored in this case.

V. Experimental Evaluation

To experimentally evaluate the usefulness of RRDtrace, we measure the accuracy of several properties when running passive monitoring applications and applying the different sampling strategies with decreasing sampling rates in a trace with real traffic. Our evaluation has three main objectives: First, to compare RRDtrace with uniform and constant sampling when both approaches reduce equally the size of the trace. Moreover, we aim to study how the three different sampling strategies with reducing sampling rates affect the accuracy on inferring traffic's properties from the RRDtrace. Finally, we examine how the accuracy is reduced across the retention time, as sampling rates are getting smaller, for different properties and sampling strategies.

We used a full payload packet trace captured during one hour at the access link that connects an educational network with thousands of hosts to the Internet. The trace contains 73,162,723 packets, corresponding to 1,728,878 different flows, totalling about 46 GB in size.

In the first set of experiments, we compare RRDtrace with the three sampling strategies which use constant sampling rate, when all the approaches reduce the size of the trace to 10% of its original size, i.e., to $S = 4.6$ GB. Thus, we applied to the original trace packet and flow sampling with 10% sampling rate and per-flow cutoff of 74 packets per flow, which all resulted to 10% of the original trace's size. In RRDtrace, we used as t_0 the most recent $1/20$ interval of the trace. In this way, RRDtrace assigned the half of the available storage, $S/2$, to this interval, selecting all the packets from it. The next two more recent $1/20$ intervals of the trace were assigned $S/4$ storage, resulting to the selection of 25% of the packets during these intervals. For the four next intervals, 6.25% sampling rate was performed, and so forth. We tried all the sampling strategies with RRDtrace and we present results from the strategy that was found to perform better with each estimated property. The produced trace was always close to 4.6 GB.We report the accuracy of each measured property separately for each of the $1/20$ intervals of the trace, to compare the different approaches with RRDtrace.

For the second set of experiments, we applied packet sampling, flow sampling and per-flow cutoff to the original trace using sampling rates from 1 to $1/4096$, resulting in multiple sampled traces. For packet and flow sampling, where packets are selected in a random way, we produced 20 traces for each case and we present the average values. Thus, for each sampling rate we created traces with each strategy. Setting $t_0 = 1$ day, for each past day we plot the value inferred from the traces with the corresponding sampling rate. For instance, $1/4096$ sampling rate corresponds to 64–127 days ago. In sampling rates below $1/256$, the respective per-flow cutoff becomes 1 packet per flow, which means that this strategy cannot be applied in very low sampling rates.

We first evaluate the accuracy on estimating flow statistics, like the original flow size distribution and average flow size. Then, we examine the accuracy on inferring the distribution of traffic and flows among the applications that generate them, using the Appmon tool [5]. Snort NIDS [6] was used to examine if the percentage of hosts and flows that produce security alerts can be inferred from the sampled traces. The accuracy for each property is measured by comparing the value inferred from each sampled trace with the real value from the unsampled trace. In the remaining of this section, we present the evaluation for each property separately.

A. Flow Size Distribution

Flow size distribution is a useful metric for traffic engineering, traffic classification, anomaly detection techniques and for studying how network traffic changes over the time. We examine the accuracy on inferring the average and mean flow size and the original distribution of flow sizes using sampled traces.

Figure 3 compares the accuracy on the estimation of average flow size using RRDtrace and sampling with constant rate for the reduction of the trace's size to 10% of its original size. RRDtrace used flow sampling, that is the best choice for inferring flow statistics, with adaptive sampling rate to retain more packets from the first parts of the trace and result also to 10% of the original trace's size. Packet and flow sampling used 10% sampling rate, while 74 packets per-flow resulted to the same reduction in the trace's size. We plot the accuracy of the average flow size estimation separately from the most recent to the older $1/20$ time interval of the original trace, by comparing with the actual value from the respective interval in the unsampled trace.

During the most recent interval RRDtrace retains all the packets, thus it is 100% accurate. For the next two intervals RRDtrace performs 25% flow sampling, so it is more accurate than 10% flow sampling. In the next four intervals, RRDtrace uses 6.25% sampling rate and its accuracy remains close to 10% flow sampling. Overall, compared with constant flow sampling, for the three more recent intervals RRDtrace is more accurate, for the next four intervals with similar accuracy and in the older intervals flow sampling outperforms RRDtrace from 5% up to mostly 20%. If we need to further reduce the storage size, e.g., to 1% of the original size using RRDtrace and 1% sampling rate, RRDtrace will be more accurate for a longer time period. When RRDtrace is used for live traffic recording, the dynamic storage re-assignment is the only way to retain data for arbitrary long periods, since sampling with constant rate will have limited retention using fixed-size storage.

Figure 4 compares the accuracy of the three different sampling strategies in RRDtrace for estimating average flow size. Flow sampling is always more accurate than packet sampling and per-flow cutoff, which are not effective strategies for the inference of flow sizes. Using flow sampling, average flow size is accurately estimated for a few days period. For the past two days, the estimation is 96.5% accurate with sampling rate 25%. RRDtrace slightly overestimates the average flow size, due to more possibilities for the selection of very large flows. For 4–7 days ago, the estimation is 94.7% accurate. RRDtrace tends to

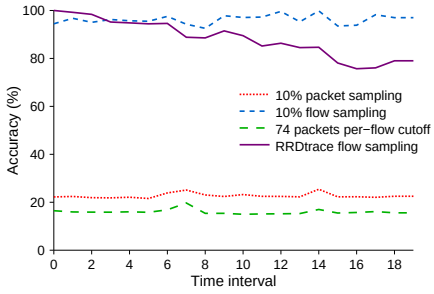

Fig. 3. Accuracy of average flow size estimation using RRDtrace and a constant 10% sampling rate for equal reduction in trace size.

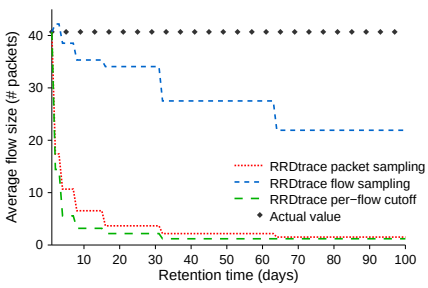

Fig. 4. Average flow size estimation using RRDtrace with different sampling strategies.

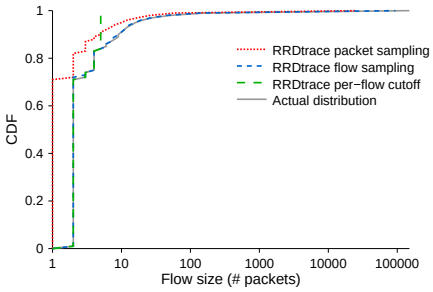

Fig. 5. Flow size distribution for 8–15 days ago, with an $1/64$ sampling rate and 5 packets per-flow cutoff.

underestimate the average flow size for lower sampling rates. Due to the heavy tailed distribution of flow sizes, there is a higher possibility for small flows to be randomly selected in low sampling rates than large flows. Moreover, very large flows cannot be selected in low sampling rates due to the storage size limitation. Up to one month ago, average flow size is estimated with 83.7% accuracy.

On the other hand, flow size distribution can be accurately estimated using flow sampling with low sampling rates. Figure 5 shows the cumulative distribution of flow sizes for packet and flow sampling with $1/64$ sampling rate and for 5 packets per-flow cutoff. Packet sampling is not accurate, since many small flows are completely lost. Per-flow cutoff strategy can estimate correctly the size of flows up to 5 packets in this case, according to the cutoff limit. The rest of the flows are considered with 5 packets size and there is no clue for their actual size. Flow sampling is accurate even with $1/4096$ sampling rate. Thus, in flow size distribution property the accuracy does not depend on the sampling rate for flow sampling. Flow sampling and per-flow cutoff estimate correctly the mean flow size, that is 2 packets per flow, while packet sampling incorrectly estimate it as one packet per flow.

While per-flow cutoff cannot accurately estimate the flow sizes, it accurately estimates the actual number of flows, since it retains at least one packet from each flow. Our flow sampling strategy can provide a less accurate estimation of the actual number of flows. If we had chosen to select exactly $s \times M$ flows from an interval with M total flows for a sampling rate s, we could infer the actual number of flows by multiplying with s the flows found in the sampled trace. Since this strategy does not always reduce the storage by s, we chose to select the number of flows with the desirable reduction in storage. Even so, we observe that for high sampling rates the chosen flow sampling strategy selects about $s \times M$ flows, so it can infer the actual number of flows. For low rates, it tends to select more than $s \times M$ flows and thus overestimates the actual number of flows up to two times in $1/4096$ sampling rate.

B. Per-Application Traffic Classification

The next property we examine is the classification of network traffic and flows to the applications that generate them. We aim to infer the percentage of traffic and flows that each application contributes to the total traffic and flows in the network. For these measurements we ran Appmon with our sampled traces. Appmon classifies flows and traffic into applications using both port-based classification and deep packet inspection to identify peer-to-peer and multimedia applications that use dynamically allocated port numbers, based on application specific signatures. For instance, Web traffic is all the packets from/to port 80, except from peer-to-peer packets masqueraded as Web packets. A flow is classified as BitTorrent flow when a packet of the flow, usually the first, contains a BitTorrent protocol-specific string. Specifically, the Peer Wire Protocol in BitTorrent establishes a handshake using well known keywords in the first packets. BitTorrent traffic is all the packets that belong to a flow classified as BitTorrent. We present the results for the two most popular applications found in the trace, Web and BitTorrent.

In Figures 6 and 9 we compare the accuracy of RRDtrace with the accuracy of 10% packet and flow sampling and 74 packets per-flow cutoff on estimating the percentages of Web and BitTorrent traffic respectively. In case of Web traffic percentage, packet sampling is clearly the most accurate strategy, due to the simple port-based classification. However, packet sampling significantly affects the detection of BitTorrent traffic, so flow sampling is the most accurate approach in this case. Comparing RRDtrace with constant 10% flow sampling in Figure 9, we observe the effect of sampling rate adaptation in RRDtrace algorithm. For the three most recent intervals RRDtrace is clearly more accurate, for the next four intervals almost equal and for the rest of the trace provides less but close accuracy compared with constant flow sampling.

Figures 7 and 10 compare the different sampling strategies across the retention time for Web and BitTorrent traffic percentage estimation respectively. We observe that packet sampling provides very accurate estimates of the Web traffic percentage regardless of how old the data are, i.e., how low the sampling rate is. Thus, packet sampling fits well with the simple port-based traffic classification. However, packet sampling cannot estimate accurately the percentage of BitTorrent traffic even for recent traffic with higher sampling rates. This is because packet sampling affects significantly the detection

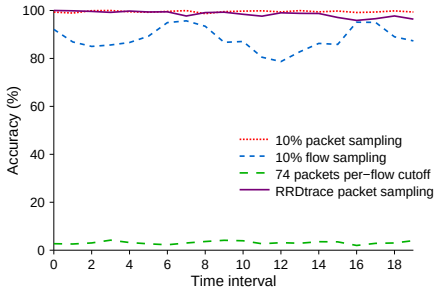

Fig. 6. Accuracy of Web traffic percentage estimation using RRDtrace and a constant 10% sampling rate for equal reduction in trace size.

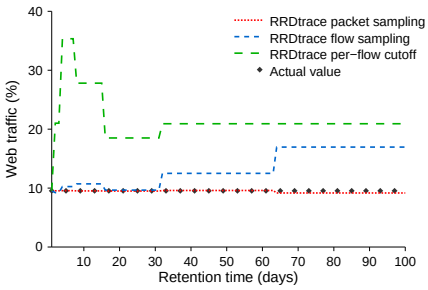

Fig. 7. Percentage of Web traffic using RRDtrace with different sampling strategies.

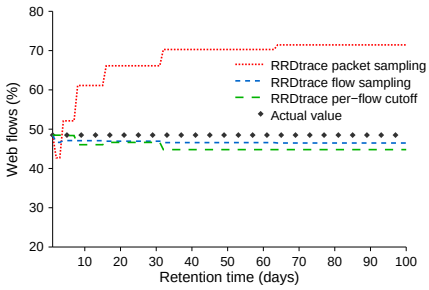

Fig. 8. Percentage of Web flows out of the classified flows.

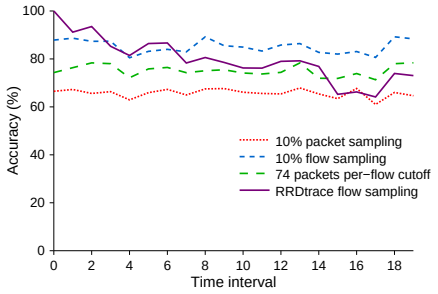

Fig. 9. Accuracy of BitTorrent traffic percentage estimation using RRDtrace and a constant 10% sampling rate for equal reduction in trace size.

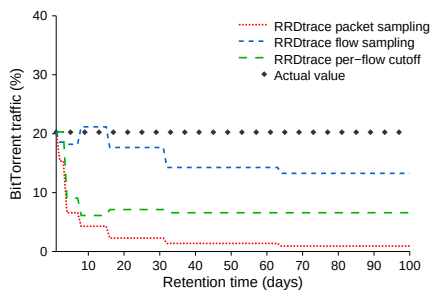

Fig. 10. Percentage of BitTorrent traffic using RRDtrace with different sampling strategies.

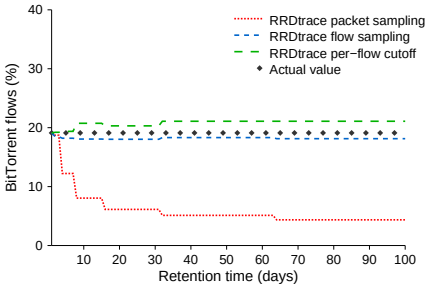

Fig. 11. Percentage of BitTorrent flows out of the classified flows.

of a BitTorrent flow. Since packets are randomly selected, the packet which contains the BitTorrent keyword may be missed. As a consequence, all the selected packets from this flow will not be classified as BitTorrent packets, leading to significant error in the estimation's accuracy.

On the other hand, flow sampling is the most accurate approach for estimating the percentage of BitTorrent traffic. In flow sampling, all packets from a selected flow are present, so the flow-based classification process is not affected. Thus, it can estimate the percentage of BitTorrent traffic till 30 days ago with more than 87% accuracy. For the same period, it can estimate the Web traffic percentage with accuracy 98.75%. The decreasing sampling rates affect the accuracy of flow sampling. While it has good accuracy up to 30 days ago, for older time periods it tends to overestimate Web traffic and underestimate the BitTorrent traffic percentage.

The third packet selection strategy, based on a per-flow cutoff, cannot account correctly the percentage of traffic for each application. While most of the traffic can be successfully classified by the first packets of the flow, the cutoff affects non-uniformly the traffic volume stored from each application. For instance, BitTorrent has usually large flows which are highly affected from the cutoff, resulting to an underestimation of the BitTorrent traffic percentage, as we observe in Figure 10. On the other hand, Web flows are typically smaller and thus less affected, leading to an overestimation of Web traffic.

However, per-flow cutoff can accurately estimate the number of Web and BitTorrent flows, even if it cannot infer the correct percentages over the total traffic. Even with one packet per flow, BitTorrent flows can be usually detected. Figure 8 shows the percentage of Web flows and Figure 11 the percentage of BitTorrent flows out of the flows that were successfully classified. We observe that both flow sampling and per-flow cutoff can accurately estimate the percentage of flows for each application for arbitrary low sampling rates, with flow sampling be slightly more accurate. While packet sampling can estimate successfully the Web traffic percentage, it cannot estimate correctly the number of Web flows.

C. Estimation of Malicious Population

Network traffic stored in sampled traces can provide some information about past networking attacks, suspicious activities and malicious hosts. Instead of trying to infer the actual attacks that happened in the past, we focus on estimating the percentage of hosts and flows that generates security alerts and how this percentage changes over the time. We ran Snort NIDS in the unsampled and sampled traces with each sampling strategy and sampling rate, aiming to measure the accuracy on estimating these percentages. We consider the source IP addresses of packets that produce Snort alerts as malicious hosts. The percentage of malicious hosts is estimated by dividing the number of unique malicious hosts with the total number of hosts that are present in each sampled trace. We also ran Snort with the reduced trace in 10% of its original size

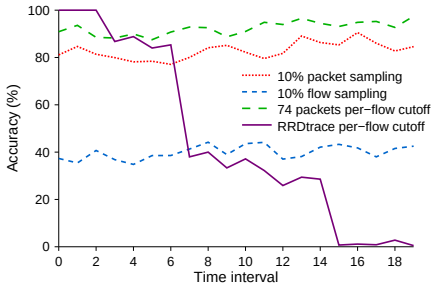

Fig. 12. Accuracy of malicious hosts estimation using RRDtrace and a constant 10% sampling rate for equal reduction in trace size.

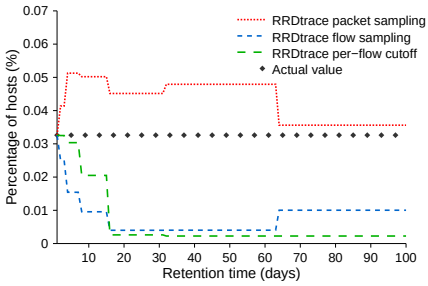

Fig. 13. Percentage of hosts that trigger security alerts.

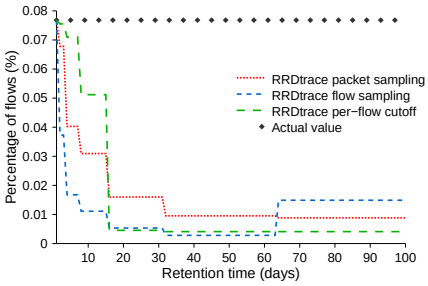

Fig. 14. Percentage of flows that trigger security alerts.

with RRDtrace and constant sampling techniques and compare their accuracy on the estimated percentage of malicious hosts.

We observe that with a per-flow cutoff of 74 packets, 84% of the alerts that were triggered in the original trace are also detected. For security applications, per-flow cutoff is a good choice for data reduction since a large class of attacks is detected in the beginning of the flows. For instance, network service probes, brute force login attempts and code-injection attacks usually appear in the first few hundred packets of a network flow. Moreover, due to the heavy tailed distribution of flow sizes, the 74-packet per-flow cutoff affects only 1.6% of the flows that contribute most of the traffic, resulting to a reduction rate of 90%.

Figure 12 presents the results for the reduced trace. Per-flow cutoff strategy has the best accuracy in estimation of the malicious host percentage. Packet sampling has better accuracy than flow sampling on the malicious host estimation. Using the trace sampled with 10% packet sampling, Snort finds significantly more attacks than with the trace produced with 10% flow sampling, 18% and 3% of the actual alerts respectively. Per-flow cutoff strategy retains all the hosts that were present in the original trace, while packet sampling only a subset of them (27.5%). Thus, the accuracy of the malicious hosts percentage in case of packet sampling is close to the accuracy when using the per-flow cutoff.

Flow sampling is not a good choice for this property. Therefore, for estimating the malicious hosts percentage we use the per-flow cutoff strategy with RRDtrace, which dynamically adapts the cutoff to store more packets per flow for the recent time intervals. In the three more recent intervals RRDtrace has 100% accuracy, since the cutoff of 2754 packets per flow that is applied in second and third intervals does not affect the malicious hosts detection. After the seventh interval, the accuracy of RRDtrace with per-flow cutoff degrades significantly, since the 5 packets per flow cutoff results to less malicious hosts be detected.

Figure 13 shows the effect of sampling rates on each strategy when estimating the percentage of malicious hosts. Per-flow cutoff is very accurate for 7 days ago and has reasonable accuracy till 15 days in the past. For older traffic, the lower sampling rates affect significantly its accuracy, leading to underestimation of malicious hosts percentage. This happens because this strategy retains all the hosts in the trace but less packets from each one, so less attacks and malicious hosts will be detected at lower sampling rates, resulting to a reduced percentage. On the other hand, packet sampling overestimates the percentage of malicious hosts and its accuracy is not affected by decreasing sampling rates. With the reduction of sampling rate, both the number of detected malicious hosts and the number of total hosts in the sampled trace are reduced. Therefore, while per-flow cutoff is more accurate at high sampling rates, at lower sampling rates packet sampling provides best accuracy and should be preferred.

Figure 14 presents the percentage of flows that trigger alerts in Snort out of the total flows found in the sampled traces. As we expected, per-flow cutoff provides the most accurate estimations for the last 15 days, but for older time periods it degrades significantly. All the sampling strategies are highly affected by the reducing sampling rates in this case. With packet sampling Snort finds a reasonable number of alerts and malicious flows, but the total number of flows found in the sampled trace is not proportional with the sampling rate. While flow sampling selects the proper amount of flows, less alerts are found in the produced traces compared with packet sampling and the percentage of malicious flows is underestimated.

VI. Conclusion

Recording raw network traffic for long-term periods is extremely useful for a multitude of monitoring and security applications. The high volumes of network traffic highlight the need for data reduction and optimized traffic storage systems. In this paper we present RRDtrace, a technique that enables storing raw network packets in fixed-size disk space for arbitrary long periods, while retaining more detailed information for most recent traffic. RRDtrace dynamically reduces storage space as traffic ages using three alternative sampling strategies: packet sampling, flow sampling, and per-flow cutoff.

We experimentally evaluated RRDtrace by measuring the accuracy of flow size distribution estimation, traffic classification, and malicious hosts detection across the retention period using real traffic. Our main findings are the following:

1) When RRDtrace is used offline to reduce the size of a trace, it provides higher accuracy for the most recent part and close accuracy for the rest, compared to constant sampling with the same effect in trace size. When RRDtrace is used for live recording, it can store packets for arbitrary long periods based on the dynamic storage reduction, while constant sampling has limited retention.

2) Some properties can be accurately inferred regardless of how old the traffic is, i.e., using arbitrary low sampling rates. Such properties include flow size distribution using flow sampling, the percentage of Web and BitTorrent flows using flow sampling or a per-flow cutoff, and the percentage of Web traffic using packet sampling.

3) In contrast, other properties are highly affected by sampling rate and can be accurately inferred only in recent periods. Such properties include average flow size, percentage of BitTorrent traffic, and percentage of hosts and flows that trigger security alerts.

4) Flow sampling is overall the most robust technique for flow statistics and traffic classification inference, but it performs poorly in estimation of malicious population. Per-flow cutoff strategy can estimate the actual number of flows and detect more attacks. However, it is not able to infer flow size and cannot be used with low sampling rates. Packet sampling can estimate very accurately the percentage of Web traffic but not BitTorrent traffic, since it highly affects the corresponding detection algorithm.

ACKNOWLEDGMENTS

This work was supported in part by the Marie Curie FP7-PEOPLE-2009-IOF project MALCODE. Antonis Papadogiannakis and Evangelos Markatos are also with the University of Crete. Part of this work was done while Michalis Polychronakis was at FORTH-ICS.

REFERENCES

[1] A. Turner, "Tcpreplay," http://tcpreplay.synfin.net/trac/.

[2] J. Micheel, H.-W. Braun, and I. Graham, "Storage and bandwidth requirements for passive internet header traces," in *Proceedings of the Workshop on Network-Related Data Management*, 2001.

[3] M. Grossglauser and J. Rexford, "Passive traffic measurement for IP operations," in *The Internet as a Large-Scale Complex System*, 2005, pp. 91–120.

[4] T. Oetiker, "Rrdtool," http://oss.oetiker.ch/rrdtool/.

[5] D. Antoniades, M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ubik, and A. Oslebo, "Appmon: An application for accurate per application traffic characterization," in *Proceedings of IST Broadband Europe 2006 Conference*, December 2006.

[6] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *Proceedings of the 1999 USENIX LISA Systems Administration Conference*, November 1999. [Online]. Available: http://www.snort.org

[7] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Computer Networks*, 1998, pp. 2435–2463.

[8] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, "Enriching network security analysis with time travel," in *SIGCOMM '08: Proceedings of the 2008 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, August 2008, pp. 183–194.

[9] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: existing techniques and new directions," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2002, pp. 161–174.

[10] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein, "Enabling real-time querying of live and historical stream data," in *SSDBM '07: Proceedings of the 19th International Conference on Scientific and Statistical Database Management*. Washington, DC, USA: IEEE Computer Society, 2007, p. 28.

[11] S. Chandrasekaran and M. Franklin, "Remembrance of streams past: overload-sensitive management of archived streams," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 348–359.

[12] Cisco Systems, "Cisco IOS Netflow," http://www.cisco.com/warp/public/732/netflow/.

[13] E. Cooke, A. Myrick, D. Rusek, and F. Jahanian, "Resource-Aware Multi-Format Network Security Data Storage," in *Proceedings of the SIGCOMM Workshop on Large Scale Attack Defense (LSAD'2006)*, September 2006, pp. 177–184.

[14] M. Ponec, P. Giura, H. Brönnimann, and J. Wein, "Highly efficient techniques for network forensics," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 150–160.

[15] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 87–95, 2000.

[16] E. Anderson and M. Arlitt, "Full Packet Capture and Offline Analysis on 1 and 10 Gb/s Networks," HP Labs, Tech. Rep., 2006.

[17] P. J. Desnoyers and P. Shenoy, "Hyperion: high volume stream archival for retrospective querying," in *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–14.

[18] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: a stream database for network applications," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2003.

[19] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2004, pp. 245–256.

[20] IETF, "Packet sampling working group cgarter," http://www.ietf.org/html.charters/psamp-charter.html.

[21] J. Drobisz and K. J. Christensen, "Adaptive sampling methods to determine network traffic statistics including the hurst parameter," in *LCN '98: Proceedings of the 23rd Annual IEEE Conference on Local Computer Networks*. Washington, DC, USA: IEEE Computer Society, 1998, p. 238.

[22] B.-Y. Choi, J. Park, and Z.-L. Zhang, "Adaptive random sampling for load change detection," *SIGMETRICS*, vol. 30, no. 1, pp. 272–273, 2002.

[23] E. A. Hernandez, M. C. Chidester, and A. D. George, "Adaptive sampling for network management," *J. Network and Systems Management*, vol. 9, no. 4, pp. 409–434, 2001.

[24] N. Duffield, C. Lund, and M. Thorup, "Flow sampling under hard resource constraints," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 85–96, 2004.

[25] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proceedings of the 6th ACM conference on Internet measurement (IMC)*, 2006, pp. 159–164.

[26] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proceedings of the 6th ACM conference on Internet measurement (IMC)*, 2006, pp. 165–176.

[27] N. Duffield, C. Lund, and M. Thorup, "Charging from sampled network usage," in *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. New York, NY, USA: ACM, 2001, pp. 245–256.

[28] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

[29] N. Hohn and D. Veitch, "Inverting sampled traffic," in *In Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003, pp. 222–233.

[30] N. Duffield, C. Lund, and M. Thorup, "Properties and prediction of flow statistics from sampled packet streams," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. New York, NY, USA: ACM, 2002, pp. 159–171.

[31] S. McCanne, C. Leres, and V. Jacobson, "libpcap," lawrence Berkeley Laboratory, Berkeley, CA. (software available from http://www.tcpdump.org/).